Javascript导学

xinqiu

Published
with GitBook

目錄

介紹 0
基础 1
注释 1.1
变量 1.2
数据类型 1.3
等式 1.4
数字 2
创建 2.1
基本运算符 2.2
高级运算符 2.3
字符串 3
创建 3.1
连接 3.2
长度 3.3
条件语句 4
If 4.1
Else 4.2
比较 4.3
条件连接 4.4
数组 5
索引 5.1
长度 5.2
循环 6
For 6.1
While 6.2
Do...While 6.3
函数 7

- 声明 7.1
- 高阶函数 7.2
- 对象 8
  - 创建 8.1
  - 属性 8.2
  - 可变性 8.3
  - 引用 8.4
  - 原型 8.5
  - 销毁 8.6
  - 枚举 8.7
  - 全局化 8.8

Javascript 导学

这本书将教你Javascript编程的相关基础。不管你是菜鸟还是一个有经验的程序猿，这本书是为任何想学习Javascript编程的人而准备的。

JavaScript (简称 *JS*)是一种能够让网页响应用户交互的编程语言。它产生于1995年，现已是当下最火并且使用最广泛的语言之一。

Note: 这本开源的书来自 GitBook的Learn Javascript。

自我说明: 这是我第一次进行开源翻译项目，现在是一个人，也欢迎有其他人加入这个项目。还希望大家指出翻译错误的地方，有问题可以在GitHub中提出。 最后，感谢本书作者Gitbook公司提供这套教程。

编程基础

在第一章，我们将学习Javascript的编程基础。

编程其实就是敲代码。就像一本书是包含章节，段落，句子，短语，单词最终由字母组成的一样，程序也能被切分成一块块更小的部分。对于程序，最重要的声明。声明等同于书中的句子。单独看，句子有结构和议题。但脱离了上下文，它却没有该有的意义。

声明，即非正式的(广泛的)被认为就是一行代码。因为声明通常被写在同一行。正是这样，程序代码通常从上到下，从左往右读。你可能会惊讶什么是代码(也就是源代码)。这是个广泛的术语，指的可以是整个程序也可以是其中最小的部分。因此，一行代码简单的说就是一行你的程序。

举个例子：

```
var hello = "Hello";
var world = "World";

// Message equals "Hello World"
var message = hello + " " + world;
```

这段代码能被 解释器 执行，并且按照正确的顺序来执行。

注释

注释不会被解释执行，注释被用来将代码解释给其他程序员看，或者是为了对代码作用的简单描述。因此提高代码的可读性。

在Javascript中，可以使用两种方式进行注释:

- 单行注释用 `//` :

```
// 这是注释，将被解释器忽略
var a = "this is a variable defined in a statement";
```

- 多行注释用 `/*` 开始，用 `*/` 结尾:

```
/*
这是多行注释，
将被解释器忽略
*/
var a = "this is a variable defined in a statement";
```

Exercise

将作者的话变成注释

```
Mark me as a comment
or I'll throw an error
```

变量

真正理解编程的第一步是回顾代数知识。如果你记得上学时曾学的代数，起初应该如下。

```
3 + 5 = 8
```

你开始执行含有未知数的计算，如下x代表未知数：

```
3 + x = 8
```

通过移项可以计算出x：

```
x = 8 - 3
-> x = 5
```

当你引入不止一个更灵活的的条件 -- 你正在使用变量:

```
x + y = 8
```

你可以改变x和y的值，表达式依然成立:

```
x = 4
y = 4
```

或

```
x = 3
y = 5
```

编程语言都是如此。在编程中，变量是可改变值的容器。变量可以保存各种类型的值或计算结果。变量由变量名、值构成，通过中间的等号(=)将他们分开。变量名可以是任何字母或者单词，但要记住其中有一些限制，比如某些是有特殊功能的关键

词。

我们来看看这是如何在Javascript中工作的，以下代码定义了两个变量，计算两个之和将其保存在第三个变量中。

```
var x = 5;
var y = 6;
var result = x + y;
```

数据类型

计算机很复杂并且可以使用更复杂的变量而不仅仅只是数字。数据类型就出现了。变量的出现伴随着几种类型，不同的语言支持不同的类型。

最常见的类型有:

- 数字
  - 浮点数: 一个数字, 例如 1.21323, 4, -33.5, 100004 或 0.123
  - 整数: 一个数字例如 1, 12, -33, 140 不可以是 1.233
- 字符串: 一行文本比如 "boat", "elephant" or "damn, you are tall!"
- 布尔: 只能表示真(true)或假(false)
- 数组: 一个数值的集合比如: 1,2,3,4,'I am bored now'
- 对象: 一种更复杂的对象的表现
- **null**: 一个空值变量，它可以为各种可能的类型。
- **undefined**: 一个未定义值，产生于当你使用一个对象的不存在属性或一个未赋值的变量时。

JavaScript是一种 “弱类型” 语言, 这意味着你不需要明确的声明变量的数据类型。你只需用 `var` 关键词来暗示你正在声明一个变量，解释器会从上下文(和引号)得出你用的是什么数据类型,

Exercise

用关键词 `var` 创建一个名为 `a` 的变量.

```

```

等式

程序猿经常需要将等价的值关联起来。这时需要用到等号。

最基础的等性运算符是 `==` 。这个运算符可以判断两个变量是否相等，即使不是相同类型。

举个例子，假设:

```
var foo = 42;
var bar = 42;
var baz = "42";
var qux = "life";
```

`foo == bar` 将判定为 `true` ，而 `baz == qux` 将判定为 `false` 。然而，尽管 `foo` 和 `baz` 是不同类型， `foo == baz` 也将判定为 `true` 。 `==` 等性运算符在判断等价性之前会尝试强制将操作数转为相同类型。这与 `===` 全等运算符不同。

`===` 全等运算符判断两个变量是否类型和值 都 相等。在这种情况下， `foo === bar` 仍然是 `true` , 而 `foo === baz` 将为 `false` . `baz === qux` 仍为 `false` .

数字

JavaScript的数字只有一种类型 - 64位浮点数。这类似于Java的 'double'。不同于其他编程语言，这里不需要区分整数类型，所以 1 和 1.0 是相同值。

在这章节，我们将学习如何创建数字和使用运算符(比如加减)。

创建

创建一个数字很容易，创建任何类型都只需要使用关键词 `var` 。

数字可以被创建来自一个常量：

```
// 这是浮点型:
var a = 1.2;

// 这是整型:
var b = 10;
```

或来自另一个变量：

```
var a = 2;
var b = a;
```

Exercise

创建一个值为 `10` 的变量 `x`， 创建一个值等于 `a` 的变量 `y`。

```
var a = 11;
```

基本运算符

你可以对数字使用一些简单的数学运算符比如：

- 加: `c = a + b`
- 减: `c = a - b`
- 乘: `c = a * b`
- 除: `c = a / b`

你可以像在数学中一样，使用括号分隔进行分隔比如： `c = (a / b) + d`

Exercise

创建一个变量 `x`，它的值为`a` 和 `b` 之和再被 `c` 除，最后乘上 `d`.

```
var a = 2034547;
var b = 1.567;
var c = 6758.768;
var d = 45084;

var x =
```

高级运算符

一些高级的运算符可以这样用，比如：

- 求余 (除法的余数): `x = y % 2`
- 累加: 让 a = 5
  - `c = a++`, 结果: c = 5 和 a = 6
  - `c = ++a`, 结果: c = 6 和 a = 6
- 递减: 让 a = 5
  - `c = a--`, 结果: c = 5 和 a = 4
  - `c = --a`, 结果: c = 4 和 a = 4

Exercise

定义一个变量 `c` 作为自减变量 `x` 对3的模。

```
var x = 10;

var c =
```

字符串

JavaScript的字符串与其他高级语言字符串的实现类似。这表示文本基于消息和数据。

在这章节将涉及一些基础。关于如何创建新的字符串和常见的一些字符串处理。 以下是是一个例子：

```
"Hello World"
```

创建

你可以在Javascript中，通过单引号或双引号定义字符串：

```
// 可以用单引号
var str = 'Our lovely string';

// 也可以用双引号
var otherStr = "Another nice string";
```

在Javascript中，字符串可以使用UTF-8编码：

```
"中文 español English हिन्दी العربية português বাংলা русский 日本語 □□□□
```

注意: 字符串不能进行减，乘或除的运算。

Exercise

创建一个名为 `str` 的变量，值为 `"abc"` 。

连接

连接同时涉及两个或以上字符串，创建了一个组合这些原始数据的更长的字符串。在Javascript中使用 **+** 运算符。

```
var bigStr = 'Hi ' + 'JS strings are nice ' + 'and ' + 'easy to add
```

Exercise

连接不同的名字，这样变量 `fullName` 就包含了John的全名。

```
var firstName = "John";
var lastName = "Smith";

var fullName =
```

长度

在Javascript中通过使用 `.length` 很容易知道字符串中有多少字母。

```
// 使用 .length
var size = 'Our lovely string'.length;
```

注意: 字符串不能进行减，乘或除的运算。

Exercise

在变量 `size` 中储存 `str` 的长度。

```
var str = "Hello World";

var size =
```

条件逻辑

条件语句可以用来测试。条件判断在编程中非常重要，比如：

首先，无论程序运行使用什么数据，所有的条件都能被用来确定程序是否正常。如果盲目的相信数据，你将陷入程序出错的麻烦。如果测试有效并且所需信息格式正确，程序就不会出错，还会变得更稳定。采取这种预防措施的行为被称作防御性编程。

另一种条件判断的作用就是分支。你可能已经接触过分支图，比如填写表格时。基本上这指的是依赖if条件语句执行不同的代码分支。

在这个章节，我们将会学习Javascript中条件逻辑的基础。

If

最简单的条件判断是if语句，语法是 `if(condition){ do this … }` 。条件判断为真，才执行分支中的代码。举个字符串的例子：

```
var country = 'France';
var weather;
var food;
var currency;

if(country === 'England') {
    weather = 'horrible';
    food = 'filling';
    currency = 'pound sterling';
}

if(country === 'France') {
    weather = 'nice';
    food = 'stunning, but hardly ever vegetarian';
    currency = 'funny, small and colourful';
}

if(country === 'Germany') {
    weather = 'average';
    food = 'wurst thing ever';
    currency = 'funny, small and colourful';
}

var message = 'this is ' + country + ', the weather is ' +
        weather + ', the food is ' + food + ' and the ' +
        'currency is ' + currency;
```

注意: 条件判断可以嵌套。

Exercise

填写 `name` 的值，验证条件判断。

```
var name =

if (name === "John") {

}
```

Else

当第一个条件语句不成立时， `else` 语句将被执行。如果你想要在特殊条件下才返回一个值，这非常有效：

```
var umbrellaMandatory;

if(country === 'England'){
    umbrellaMandatory = true;
} else {
    umbrellaMandatory = false;
}
```

`else` 语句可以和另一个 `if` 语句结合。改造一下上面的例子：

```
if(country === 'England') {
    ...
} else if(country === 'France') {
    ...
} else if(country === 'Germany') {
    ...
}
```

Exercise

填写 `name` 的值，验证 `else` 语句。

```
var name =

if (name === "John") {

} else if (name === "Aaron") {
    // Valid this condition
}
```

比较运算符

把焦点放在条件判断部分：

```
if (country === "France") {
    ...
}
```

变量 `country` 后面跟着的三个等号(`===`)是条件判断部分。三个等号测试是否变量 `country` 和 `France` 值与类型(`String`)相同。你也可以用两个等号来测试，比如 `if (x == 5)` ，在 `var x = 5;` 或 `var x = "5";` 情况下都返回真。这很不一样取决于你的程序是做什么。比较推荐你经常去尝试比较三个等号(`===` 和 `!==`)和两个等号(`==` 和 `!=`)的区别。

其他条件判断的测试：

- `x > a` : is x bigger than a?
- `x < a` : is x less than a?
- `x <= a` : is x less than or equal to a?
- `x >=a` : is x greater than or equal to a?
- `x != a` : is x not a?
- `x` : does x exist?

Exercise

添加一种条件判断，如果 `x` 比5大，使变量 `a` 赋值为10。

```
var x = 6;
var a = 0;
```

逻辑比较

为了避免if-else麻烦，可以利用一种简单的逻辑比较。

```
var topper = (marks > 85) ? "YES" : "NO";
```

在上述例子中， `?` 是逻辑运算符。上述源码表示如果marks的值大于85即 `marks > 85` ，则 `topper = YES` ；否则 `topper = NO` 。基本上，如果比较条件为真，赋第一个参数的值，否则赋的二哥参数的值。

条件连接

此外，你可以用 "or” 或 “and” 语句连接不同的条件判断，可以分别的测试是否存在一个为真或同为真。

在Javascript中，“or” 可以被写成 `||` ， “and” 可以被写成 `&&` 。

比如你想要测试 x 的值是否在10到20之间，你可以用上述的方法：

```
if(x > 10 && x < 20) {
    ...
}
```

如果你想要确认 country 是 “England” 或 “Germany”：

```
if(country === 'England' || country === 'Germany') {
    ...
}
```

注意: 就像对数字的运算符，条件可以用括号来分组，比如： `if ( (name === "John" || name === "Jennifer") && country === "France")` 。

Exercise

填写两个条件让仅当name 为 `"John"` ， country 为 `"England"`, `primaryCategory` 才等于 `"E/J"` ， 仅当name 为 `"John"` 或 country 为 `"England"`， `secondaryCategory` 才等于 `"E/J"` 。

```
var name = "John";
var country = "England";
var primaryCategory, secondaryCategory;

if ( /* Fill here */ ) {
    primaryCategory = "E/J";
}
if ( /* Fill here */ ) {
    secondaryCategory = "E|J";
}
```

数组

数组是编程的基础部分。一个数组就是一系列数据。我们可以储存许多数据在一个变量中，这提高了代码可读性，让人更好理解代码。这使相关数据传递到函数中执行更简单。

数组中数据称为 元素。

这是一个简单的数组：

```
// 1, 1, 2, 3, 5 和 8 是数组中的元素
var numbers = [1, 1, 2, 3, 5, 8];
```

索引

若你有一个数组，如何访问想要的特定元素？索引出现了。一个 下标 指向数组中的一个位置。正如其他大部分语言，索引逻辑上是一个接一个，但要注意第一个数组下标是0。方括号[]被用来表示使用数组下标。

```
// 这是字符串构成的数组
var fruits = ["apple", "banana", "pineapple", "strawberry"];

// 用fruits数组中的第二个元素的值赋值给变量banana
// 记住索引(即下标)是从0开始，下标1即第二个元素
// 结果: banana = "banana"
var banana = fruits[1];
```

Exercise

定义变量使用数组的索引

```
var cars = ["Mazda", "Honda", "Chevy", "Ford"]
var honda =
var ford =
var chevy =
var mazda =
```

长度

数组有个属性称为长度，正如字面意思，这就是数组的长度。

```
var array = [1 , 2, 3];

// 结果: l = 3
var l = array.length;
```

Exercise

定义变量a为数组的长度

```
var array = [1, 1, 2, 3, 5, 8];
var l = array.length;
var a =
```

循环

循环是一种变量在其中不断改变的重复状。如果您希望一遍又一遍地运行相同的代码，并且每次的值都不同，那么使用循环是很方便的。

与其用：

```
doThing(cars[0]);
doThing(cars[1]);
doThing(cars[2]);
doThing(cars[3]);
doThing(cars[4]);
```

不如用：

```
for (var i=0; i < cars.length; i++) {
    doThing(cars[i]);
}
```

For 循环

for语句是最简单的循环形式。它的语法类似if语句，只是多了些选项：

```
for(condition; end condition; change){
    // do it, do it now
}
```

来看看如何使用for循环执行10次相同的代码：

```
for(var i = 0; i < 10; i = i + 1){
    // 执行这段代码10次
}
```

> 注意: `i = i + 1` 也可以写成 `i++`.

Exercise

使用for循环，创建一个 `message` 变量，使其值为整数0到99 (0, 1, 2, ...) 之和。

```
var message = "";
```

While 循环

只要特殊条件为真，While循环就一直执行代码块。

```
while(condition){
    // 只要条件为真就继续执行
}
```

举个例子，以下代码会一直执行，只要变量 i 小于5：

```
var i = 0, x = "";
while (i < 5) {
    x = x + "The number is " + i;
    i++;
}
```

Do/While循环是while循环的变体。这种循环将先检查条件是否为真，再执行代码块。它将重复循环只要条件为真：

```
do {
    // 代码块被执行
} while (condition);
```

注意: 要注意避免如果条件总为真导致的无限循环！

Exercise

使用while循环，创建一个 `message` 变量，保存连接的整数，仅当长度小于100。

```
var message = "";
```

Do...While 循环

do...while语句创建的循环会先执行语句直到条件判断为假。 do... while的语法如下：

```
do{
    // statement
}
while(expression) ;
```

来看看如何使用 `do...while` 循环答应小于10的数字：

```
var i = 0;
do {
    document.write(i + " ");
    i++; //  i 增加 1
} while (i < 10);
```

> 注意: `i = i + 1` 也可以写成 `i++` .

Exercise

使用do...while循环，打印小于5的数字。

```
var i = 0;
```

函数

函数是编程中功能最强大也是最重要的概念之一。

函数就像数学中的函数，有输入值即 参数 ， 返回输出值。

函数声明

如同变量，函数必须被声明。以下是声明一个函数 `double` 接受 参数 `x` 返回 x 的两倍：

```
function double(x) {
    return 2 * x;
}
```

> 注意: 上述函数可能在声明之前被引用。

函数在Javascript中也是变量；可以储存值（比如数字，字符串等等），将其作为参数传给其他函数：

```
var double = function(x) {
    return 2 * x;
};
```

> 注意: 上述函数没有可能在声明之前被引用，就像其他变量。

Exercise

声明一个 `triple` 函数，接受一个参数，返回它的三倍。

高阶函数

高阶函数是操作其他函数的函数。举个例子，一个函数可以将其他函数作为参数和/或产生一个函数作作为它的返回值。 精巧的 功能技术是非常强大的构造，可在Javascript或其他高级语言比如python，lisp等中使用。

我们来创建两个简单的函数， `add_2` 和 `double` ，和一个高阶函数 `map` 。 `map` 接受两个参数， `func` 和 `list` （它的声明因此为 `map(func,list)` ），返回一个数组。 `func` (第一个参数)作为函数将被应用到每个数组 `list` （第二个参数）的每一个元素中。

```
// 定义两个简单函数
var add_2 = function(x) {
    return x + 2;
};
var double = function(x) {
    return 2 * x;
};

//  map接受两个参数
//  func    调用函数
//  list    数组中元素值传递给 func
var map = function(func, list) {
    var output=[];              // 输出 list
    for(idx in list) {
        output.push( func(list[idx]) );
    }
    return output;
}


// 使用map将函数应用于整个输入列表
// 产生相应的输出列表
map(add_2, [5,6,7]) // => [7, 8, 9]
map(double, [5,6,7]) // => [10, 12, 14]
```

上述例子中的函数很简单。然而在作为参数传递给其他函数时，可以使用特殊组合方法构件更复杂的函数。

举个例子，如果注意到在代码中平凡使用调用方法 `map(add_2, ...)` 和 `map(double, ...)` ，可以创建两个特殊用途的列表运行组合起来的函数。使用函数封装，可以如下操作：

```
process_add_2 = function(list) {
    return map(add_2, list);
}
process_double = function(list) {
    return map(double, list);
}
process_add_2([5,6,7]) // => [7, 8, 9]
process_double([5,6,7]) // => [10, 12, 14]
```

创建一个 `buildProcessor` 函数，它将函数 `func` 作为函数，返回一个 `func` 处理程序，这个函数将列表作为输入应用到 `func` 。

```
// 函数产生一系列处理程序
var buildProcessor = function(func) {
    var process_func = function(list) {
        return map(func, list);
    }
    return process_func;
}
// 调用buildProcessor返回一个调用输入列表的函数


// 使用buildProcessor产生 add_2 process_double：
process_add_2 = buildProcessor(add_2);
process_double = buildProcessor(double);

process_add_2([5,6,7]) // => [7, 8, 9]
process_double([5,6,7]) // => [10, 12, 14]
```

看另一个例子。创建一个 `buildMultiplier` 函数，将 `x` 作为输入，然后一个对其参数 `x` 相乘的函数：

```
var buildMultiplier = function(x) {
    return function(y) {
        return x * y;
    }
}

var double = buildMultiplier(2);
var triple = buildMultiplier(3);

double(3); // => 6
triple(3); // => 9
```

Exercise

定义一个 `negate` 函数将 `add1` 作为参数，返回一个函数,这个函数通过 `add1` 返回值的相反数.(开始有点复杂了)

```
var add1 = function (x) {
    return x + 1;
};

var negate = function(func) {
    // TODO
};

// 应该返回 -6
// 因为 (5+1) * -1 = -6
negate(add1)(5);
```

对象

Javascript的原始类型是 `true` , `false` , 数字, 字符串, `null` and `undefined` 。所有其他值都是 `对象` .

Javascript对象包含成对的 `propertyName（属性名）` : `propertyValue（属性值）` 。

创建

There are two ways to create an `object` in JavaScript: 在Javascript中，有两种方法去创建一个 `对象` ：

1. 字面上

   ```
   var object = {};
     // 是的，简单是使用一对大括号！
   ```

   > 注意: 这是 推荐 的方法

2. 面向对象

   ```
   var object = new Object();
   ```

   > 注意: 这很像Java

属性

对象的属性是一对的 `propertyName(属性名)` : `propertyValue(属性值)` ，属性的名字只能是字符串。如果不是字符串，将会转换为字符串。可以在创建对象或之后初始化属性。

```
var language = {
    name: 'JavaScript',
    isSupportedByBrowsers: true,
    createdIn: 1995,
    author:{
        firstName: 'Brendan',
        lastName: 'Eich'
    },
  // 是的，对象可以嵌套！
    getAuthorFullName: function(){
        return this.author.firstName + " " + this.author.lastName;
    }
  // 是的，函数可以有值！
};
```

以下代码展示如何 获取 属性的值。

```
var variable = language.name;
  // 变量包含字符串"JavaScript"
    variable = language['name'];
  // 这行代码和上行功能一样。不同之处在于这行代码将书面化的字符串作为属性名，不这
    variable = language.newProperty;
  // 变量没定义，因为该属性没赋值。
```

以下代码展示如何添加一个新属性或改变一个存在的属性。

```
language.newProperty = 'new value';
 // 现在对象有一个新的属性。如果该属性已经存在，值将会被替换。
language['newProperty'] = 'changed value';
 // 两以上种方法都可以使用，更推荐第一种(使用`.`)。
```

可变性

对象与原始值不同之处在于对象可以改变，原始值是不可变的。

```
var myPrimitive = "first value";
    myPrimitive = "another value";
 // myPrimitive现在指向另一个字符串
var myObject = { key: "first value"};
    myObject.key = "another value";
 // myObject 指向的还是原来的对象
```

引用

对象是不会被复制的。它们的传递靠引用。

```
// 假设我有一个匹萨
var myPizza = {slices: 5};
 // 然后我和你分享它
var yourPizza = myPizza;
 // 我吃了一小片
myPizza.slices = myPizza.slices - 1;
var numberOfSlicesLeft = yourPizza.slices;
 // 我们总共有4片
 // 因为我们引用了同一块匹萨
var a = {}, b = {}, c = {};
 // a, b, 和 c 都引用不同的空对象
a = b = c = {};
 // a, b, 和 c 都引用同一个空对象
```

原型

每个对象都与对象原型关联，继承了对象原型的属性。

所有对象创建时都从字面量对象（ `{}` ）继承，且都自动链接到的 Object.prototype，Object.prototype 来自JavaScript标准。

当JavaScript解释器（在浏览器中一个模块），试图找到一个属性，它要检索，如下面的代码：

```
var adult = {age: 26},
    retrievedProperty = adult.age;
  // 看上一行
```

首先，解释器检查对象有的每个属性。例如， `adult` 只有一个自己的属性 - `age` 。但是，除此之外，实际上还有几个属性，这是继承自Object.prototype。

```
var stringRepresentation = adult.toString();
  // 变量的值为 '[object Object]'
```

`toString` 是一个 Object.prototype 的属性, 这是继承。它有一个函数，返回值为一个对象的字符串。如果希望它返回一个更有意义的东西，那么你可以将其覆盖。简单的添加一个属性到adult对象。

```
adult.toString = function(){
    return "I'm "+this.age;
}
```

如果现在调用 `toString` 函数，解释器将发现一个新的对象中的属性然后停止。

解释器解析某个属性时会先检查对象自己，再进一步检查他的原型。

要设置自己的对象为原型而不是默认的Object.prototype，你可以调用以下的 `Object.create` ：

```
var child = Object.create(adult);
 /* 通过这种方式创建的对象可以让我们轻松替换默认的Object.prototype成我们想要
 */
child.age = 8;
 /* 在此之前，child根本没有自己的年龄属性，解释器会寻找child的原型中是否有该属
注意：adult的年龄仍为26。
  */
var stringRepresentation = child.toString();
 // 值为 "I'm 8"。
 /* 注意：我们没覆盖child的toString属性，因此adult类函数不会被调用。如果adu
 */
```

`child` '的原型是 `adult` ，其原型为 `Object.prototype` 。这一系列原型被称为原型链。

销毁

`delete` 被用来从一个对象中 删除一个属性 。它将删除对象中的一个存在属性，使其不再存在于原型链中。 从一个对象中删除一个属性就是将改属性从原型中移出：

```
var adult = {age:26},
    child = Object.create(adult);
    child.age = 8;

delete child.age;
 /* 从child中删除age属性，表明这之后该属性不在被覆盖 */
var prototypeAge = child.age;
 // 26，因为该孩子没有自己的age属性。
```

枚举

`for in` 语句可以遍历对象中所有的属性。枚举包括函数和原型属性。

```
var fruit = {
    apple: 2,
    orange:5,
    pear:1
},
sentence = 'I have ',
quantity;
for (kind in fruit){
    quantity = fruit[kind];
    sentence += quantity+' '+kind+
                (quantity===1?'':'s')+
                ', ';
}
 // The following line removes the trailing coma.
sentence = sentence.substr(0,sentence.length-2)+'.';
 // I have 2 apples, 5 oranges, 1 pear.
```

全局化

如果想开发一个模块，它可以在网页上运行或也可以运行其他模块，因此你必须注意变量名是否重复。

假设我们正在卡开发一个计数器模块：

```
var myCounter = {
    number : 0,
    plusPlus : function(){
        this.number : this.number + 1;
    },
    isGreaterThanTen : function(){
        return this.number > 10;
    }
}
```

> 注意: 这个技巧通常配合闭包使用，以使来自外部的内部状态不变。

模块使用唯一一个变量名 — `myCounter` 。如果其他模块使用名字比如 `number` 或 `isGreaterThanTen` ，这样就会很安全，因为不会覆盖每个其他的值。